Announcements

• WQ6, HW6 due next Monday

• WQ7, HW7 will be out next Monday
Data Management Pipeline

Conceptual Schema

Schema designer

Application programmer

Database administrator

Physical Schema

Table representation:

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3
Transactions

• We use database transactions everyday
  – Bank $$$ transfers
  – Online shopping
  – Signing up for classes

• For this class, a transaction is a series of DB queries
  – Read / Write / Update / Delete / Insert
  – Unit of work issued by a user that is independent from others
What’s the big deal?
Challenges

- Want to execute many apps concurrently
  - All these apps read and write data to the same DB

- Simple solution: only serve one app at a time
  - What’s the problem?

- Want: multiple operations to be executed atomically over the same DBMS
What can go wrong?

• Manager: balance budgets among projects
  – Remove $10k from project A
  – Add $7k to project B
  – Add $3k to project C

• CEO: check company’s total balance
  – SELECT SUM(money) FROM budget;

• This is called a dirty / inconsistent read aka a WRITE-READ conflict
What can go wrong?

- App 1:
  SELECT inventory FROM products WHERE pid = 1

- App 2:
  UPDATE products SET inventory = 0 WHERE pid = 1

- App 1:
  SELECT inventory * price FROM products
  WHERE pid = 1

- This is known as an unrepeatable read
  aka READ-WRITE conflict
What can go wrong?

Account 1 = $100
Account 2 = $100
Total = $200

• App 1:
  – Set Account 1 = $200
  – Set Account 2 = $0

• App 2:
  – Set Account 2 = $200
  – Set Account 1 = $0

• At the end:
  – Total = $200

• App 1: Set Account 1 = $200

• App 2: Set Account 2 = $200

• App 1: Set Account 2 = $0

• App 2: Set Account 1 = $0

• At the end:
  – Total = $0

This is called the lost update aka WRITE-WRITE conflict
What can go wrong?

• Buying tickets to the next Bieber concert:
  – Fill up form with your mailing address
  – Put in debit card number
  – Click submit
  – Screen shows money deducted from your account
  – [Your browser crashes]

Lesson:
Changes to the database should be **ALL or NOTHING**
Transactions

• Collection of statements that are executed atomically (logically speaking)

BEGIN TRANSACTION
  [SQL statements]
COMMIT or
ROLLBACK (=ABORT)

If BEGIN… missing, then TXN consists of a single instruction
Transactions Demo
Serial execution

- **Definition**: A SERIAL execution of transactions is one where each transaction is executed one after another.

- **Fact**: Nothing can go wrong if the DB executes transactions serially.

- **Definition**: A SERIALIZABLE execution of transactions is one that is equivalent to a serial execution.
What we want: ACID

- **Atomic**
  - State shows either all the effects of txn, or none of them
- **Consistent**
  - Txn moves from a DBMS state where integrity holds, to another where integrity holds
    - remember integrity constraints?
- **Isolated**
  - Effect of txns is the same as txns running one after another (i.e., looks like batch mode)
- **Durable**
  - Once a txn has committed, its effects remain in the database
Atomic

• **Definition:** A transaction is ATOMIC if all its updates must happen or not at all.

• **Example:** move $100 from A to B
  - UPDATE accounts SET `bal = bal − 100` WHERE `acct = A`;
  - UPDATE accounts SET `bal = bal + 100` WHERE `acct = B`;
  
  ```
  BEGIN TRANSACTION;
  UPDATE accounts SET `bal = bal − 100` WHERE `acct = A`;
  UPDATE accounts SET `bal = bal + 100` WHERE `acct = B`;
  COMMIT;
  ```
Isolated

- **Definition** An execution ensures that txns are isolated, if the effect of each txn is as if it were the only txn running on the system.
Consistent

• Recall: integrity constraints govern how values in tables are related to each other
  – Can be enforced by the DBMS, or ensured by the app

• How consistency is achieved by the app:
  – App programmer ensures that txns only takes a consistent DB state to another consistent state
  – DB makes sure that txns are executed atomically

• Can defer checking the validity of constraints until the end of a transaction
Durable

• A transaction is durable if its effects continue to exist after the transaction and even after the program has terminated

• How?
  – By writing to disk!
Rollback transactions

• If the app gets to a state where it cannot complete the transaction successfully, execute ROLLBACK

• The DB returns to the state prior to the transaction

• What are examples of such program states?
ACID

- Atomic
- Consistent
- Isolated
- Durable

- Enjoy this in HW7!

- Again: by default each statement is its own txn
  - Unless auto-commit is off then each statement starts a new txn
Implementation of transactions

• sqlite: single lock for the entire DB
  – http://www.sqlite.org/atomiccommit.html
  – Not true for SQL Server, DB2, etc
SQLite Transactions

- **Step 1:** When txn starts: acquires a read lock (aka shared lock) (recall CSE 332?)
- **Step 2:** When txn writes: acquire a reserved lock
- **Step 3:** When txn commits:
  - First acquire a pending lock: no new read locks allowed
  - Wait until all current read locks are released
  - Acquire an exclusive lock
  - Make updates to DB on disk
  - Commit, release all locks